CSci 5271: Introduction to Computer Security

Exercise Set 5 **due: Thursday, December 4th, 2014**

Ground Rules. You may choose to complete these exercises in a group of up to three students. Each group should turn in **one** copy with the names of all group members on it. You may use any source you can find to help with this assignment but you **must** explicitly reference any source you use besides the lecture notes or textbook. An electronic (plain text or PDF) copy of your solution should be submitted on the course Moodle by 11:55pm on Thursday, December 4th.

1. Cross-site scripting variations. (30 pts) There are a lot of different kinds of cross-site scripting vulnerabilities, but for space reasons we only covered one of them (intentionally) in hands-on assignment 2. This question covers some others.

(a) Here's an excerpt from some Java code in the implementation of question 6 from the hands-on assignment:

```
public class MACCookieServlet extends GroupServlet {
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
                    throws ServletException, IOException {
        String username = req.getParameter("username");
        if (username == null)
             username = "";
        String digest_hex = ...;
        resp.setStatus(HttpServletResponse.SC_OK);
        resp.setContentType("text/html;charset=utf-8");
        resp.getWriter().print("User \"");
        resp.getWriter().print(username);
        resp.getWriter().print("\" is identified with the MAC ");
        resp.getWriter().print(digest_hex);
        resp.getWriter().println("\".");
    }
}
```

As mentioned in class, this code suffers a reflected XSS vulnerability: the `username` parameter is under the control of the untrusted user, and it is copied directly into the HTML output. So if it contained JavaScript, that code would run with the site's permissions. There's no similar problem with `digest_hex`, because it is transformed (in the omitted code) to contain only hexadecimal digits.

One way to fix this vulnerability would be to sanitize the contents of the `username` string using HTML entities; for instance, translating each "`<`" into "`<`". But maybe the programmer didn't know what library would contain a good implementation of that translation or was too lazy to implement it himself. What other simple change could you make to this code to avoid the cross-site-scripting danger?

(b) A DOM-based XSS vulnerability (also sometimes called "type-0") arises when the behavior of client-side JavaScript code allows untrusted data to be interpreted as JavaScript, or as HTML that can contain JavaScript, et cetera. The web site in hands-on assignment 2 didn't contain much JavaScript so there was not much possibility for it to have a DOM-based XSS vulnerability, but such problems are more likely to arise on sites that make sophisticated use of JavaScript. Here's a list of some JavaScript-related features that are often used on more modern sites:

- XSLT (Extensible Stylesheet Language Transformations): a functional language for transforming XML documents into other XML documents
- XHR (XMLHttpRequest): an API that lets JavaScript code make separate HTTP or HTTPS requests to a server
- JSON (JavaScript Object Notation): a text-based data interchange format that is a subset of JavaScript
- CSS (Cascading Style Sheets): a declarative language for specifying the presentation style of an HTML document

Pick one of these features and explain how, if it is not used carefully, it can lead to a DOM-based XSS vulnerability.

2. Virus Virii. (30 pts) Sam has invented a brand-new virus detector, ViruSniff, and he claims it is "100% effective" — if executable file F is a virus, then ViruSniff(F) will output "`VIRUS!!!`".

(a) Does ViruSniff's claim conflict with the undecidability of the halting problem? Why or why not? (Hint: is there another term besides "effectiveness" that describes that statistic that Sam claims is 100%? Is there a simple program that can do exactly what Sam says ViruSniff can do?)

(b) Some hackers reverse engineer ViruSniff and post its algorithm online: it turns out that ViruSniff does processor emulation of the first 10000 instructions of an executable, and then applies a fancy signature matching algorithm (that no one seems to understand) to the sequence of instructions and memory changes to decide if the program is a virus or not. Explain how to change any program that runs for at least 10001 instructions, and does not trigger the `VIRUS!!!` alert, to propagate a virus such that the altered program will also fail to trigger the alert. What does your strategy say about Sam's claim?

(c) Given your knowledge of the attack from (b), how might you enhance ViruSniff to work against the new virus-writing strategy? Evaluate the potential effect of your change on the false-positive rate.

3. Denial of Service Denial. (40 pts) Sly and Carl are really concerned about the possibility of DoS attacks against their web server program.

(a) Sly has developed a new module for his web server that he claims will prevent DoS attacks by slowing them down. In Sly's module, every incoming HTTP request is put into a queue, with a timestamp and a "delayed" bit marked as false. When it is ready to serve a request, the web server takes the first request in the queue. If the "delayed" bit is false and there are no other requests from the same IP address in the queue, it serves the request immediately. If the "delayed" bit is false and there is at least one other request from the same IP address in the queue, the "delayed" bit is set to true and the request is re-inserted at the end of the queue. If the delayed bit is set to "true," then the request is served **if** the current time is at least 1 second greater than the request timestamp, and **otherwise** the request is sent to the end of the queue again. Sly's idea is that this will allow the site to deal with requests from legitimate users in preference to DoS attack requests.

Will Sly's scheme work to prevent a DoS attack from making his web server unusable by normal users? Give a detailed explanation.

(b) Inspired by BitTorrent, Carl has a different suggestion for preventing DoS. In Carl's solution, whenever client downloads a page, he also downloads an ActiveX control that acts as a mini web server for that page and its contents only. Then when the main server starts to be overloaded, it uses HTTP redirects to point new clients to servers running on old clients. The new clients can then download the pages from old clients directly, without using any more of the main server's bandwidth.

There are some implementation challenges with Carl's scheme, such as browsers that are behind firewalls or don't support ActiveX. But let's assume these are adequately solved. How well will Carl's scheme work against attackers who want to make his site unusable?